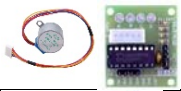
# Dokumentation Projekt "be PArd"  Version 2.0

Nina Wentz und Christian Hermes

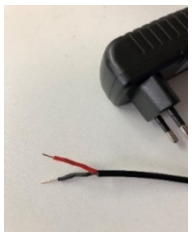
# Inhalt

# Kisteninhalt Stand Dezember 2017

| | | |
|---|---|---|
|  | 1 | WeMos D1 Mini WLAN Module, Entwicklungsboard |
|  | 1 | Breadboard Steckbrett 830 Kontakte |
|  | 1 | Micro USB Kabel Nylon 1m |
|  | 1 | 5V 3A Netzteil DC Adapter |
|  | 16 | Jumper Wire Male Male, unterschiedliche Länge und Farbe |
|  | 2 | 20cm Jumper female-female |
|  | 10 | 20cm Jumper male-female |
|  | 1 | Widerstand 100 Ohm |
|  | 4 | Widerstand 1 K Ohm |
|  | 2 | 12 x 12 x 7.3mm Push Button 4 Pins mit Kappe |
|  | 10 | (5 Farben x 50pcs), 3mm LED |
|  | 10 | Widerstand 330 Ohm |
|  | 2m | RGB LED-Band |
|  | 1 | Schrittmotor 5V mit Treiberplatine |
|  | 2 | Fotowiderstand |
|  | 1 | Achssensormodul, bi-axialer Joystick mit Taster |
|  | 1 | 128 x 64 Pixel 0,96 Zoll OLED $I^2C$ Display |
|  | 1 | Temperatur- und Luftfeuchtigkeitssensor |

# Vorbereitungen

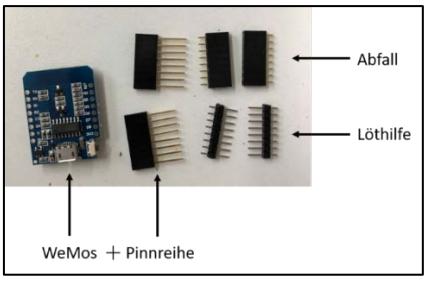
## Netzteil

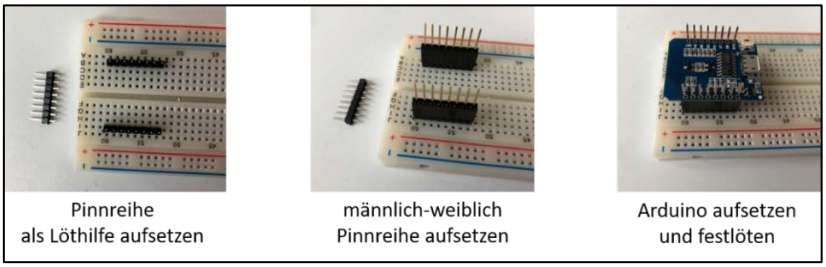


USB-Stecker abschneiden und Kabelstücke von ca. 1-2cm anlöten. Danach die Kabelenden mit Schrumpfschlauch versehen.

## WeMos D1 mini

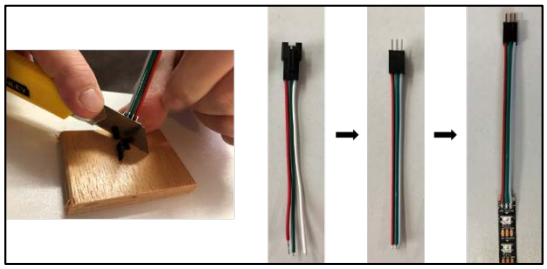
Der WeMos wird mit drei verschiedenen Pinnreihen ausgeliefert: männlich, weiblich und männlich-weiblich. Wir haben entschieden, die männlich-weiblich Pinnreihe aufzulöten, damit man den WeMos auf das Breadboard draufstecken und gleichzeitig Kabel direkt an den WeMos anschließen kann. Die männliche Pinnreihe wird als Löthilfe auf das Breadboard gesteckt. Darauf die männlich-weibliche Pinnreihe und den WeMos setzen und festlöten.




## LED-Streifen

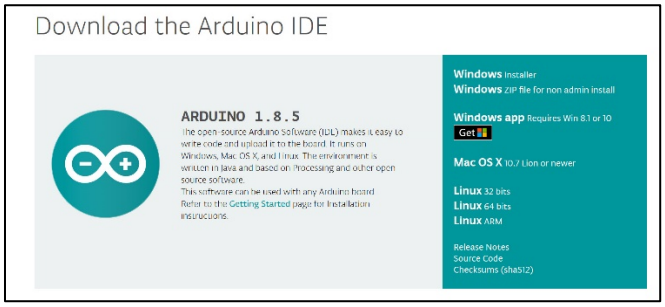
Die LED-Streifen sind standardmäßig mit Mamasteckern auf der Eingangsseite und Papasteckern auf der Ausgangsseite versehen. Beide werden abgeschnitten. Der Papastecker wird von seiner Plastikhülle befreit, damit er aufs Breadboard aufgesteckt werden kann. Dann wird er an die Eingangsseite des LED-Streifens gelötet.



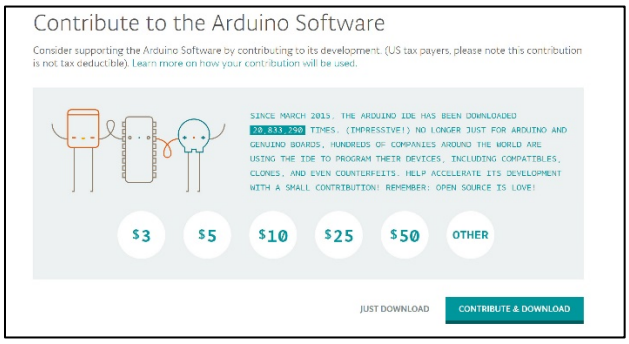
# Software

## Software Arduino installieren

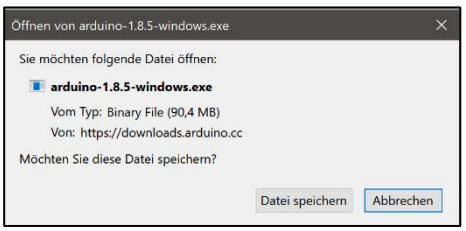
Die Arduino Software wird von der Homepage "https://www.arduino.cc/en/Main/Software" heruntergeladen. Für Windowsrechner bitte auf **Windows Installer** klicken.



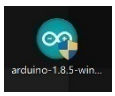

Für eine kostenlose Version bitte auf **just download** klicken.




Die Datei speichern. Voreingestellt wird die Datei unter downloads gespeichert. Je nach geänderten Voreinstellungen wird sie anders abgelegt.

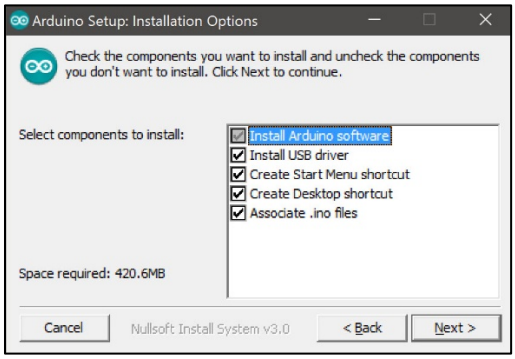



Auf die heruntergeladene exe-Datei klicken.

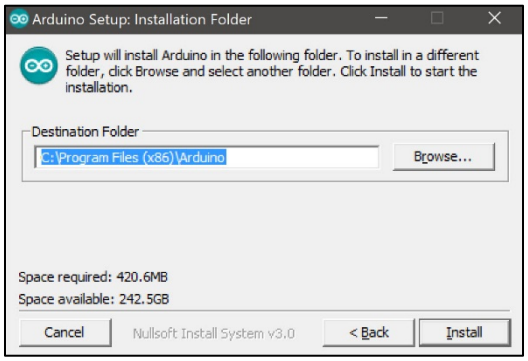


Die Lizenzvereinbarung mit **I agree** bestätigen

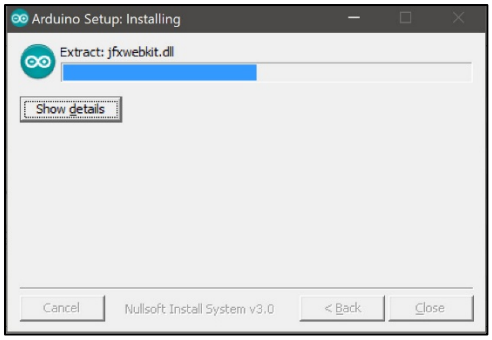

Alle Haken so belassen und mit **Next** bestätigen



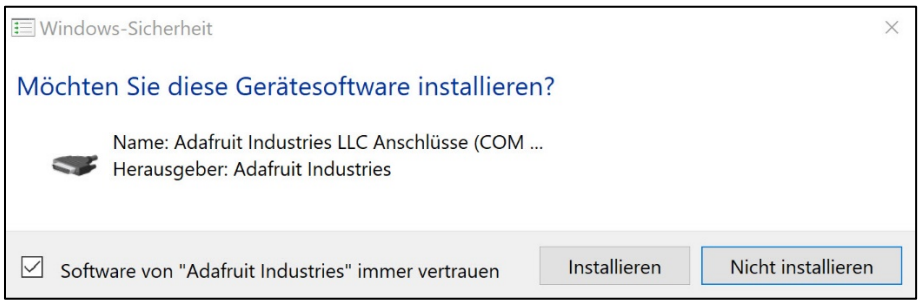

Den voreingestellten Ordner belassen und mit **Install** weitergehen



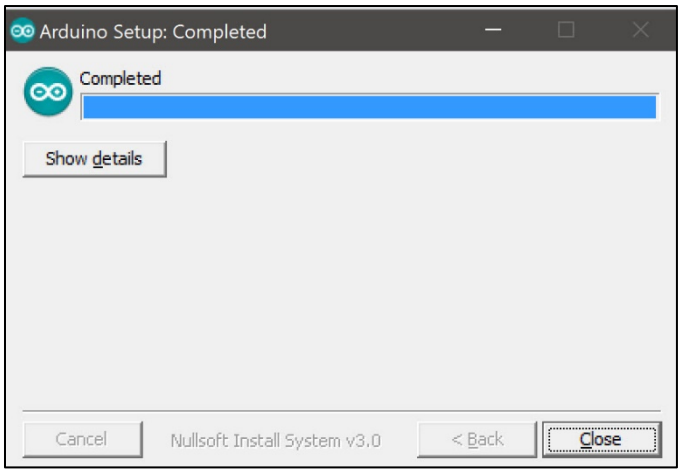

Das Programm wird nun installiert.



Eventuell fragt die Windows Sicherheit nach, ob die Gerätesoftware installiert und ihr vertraut werden soll. Dies mit **Installieren** bestätigen.




Nach Abschluss der Installation das Setup mit **Close** verlassen.

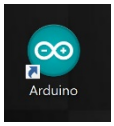



# Treiber installieren

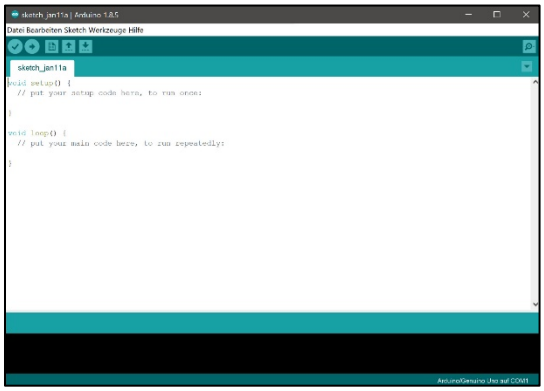
Eventuell muss ein USB-Treiber zusätzlich installiert werden, damit der WeMos erkannt wird. Der Treiber **DRIVER1_CH340** kann z.B. von der Homepage http://www.arduined.eu/ch340-windows-8-driver-download/ gedownloaded werden.

In dem heruntergeladenen Ordner "DRIVER1_CH340" den Ordner "DRIVER1" öffnen. Die Anwendung "SETUP.EXE" (evtl. wird nur "SETUP" angezeigt) öffnen und den Treiber installieren.

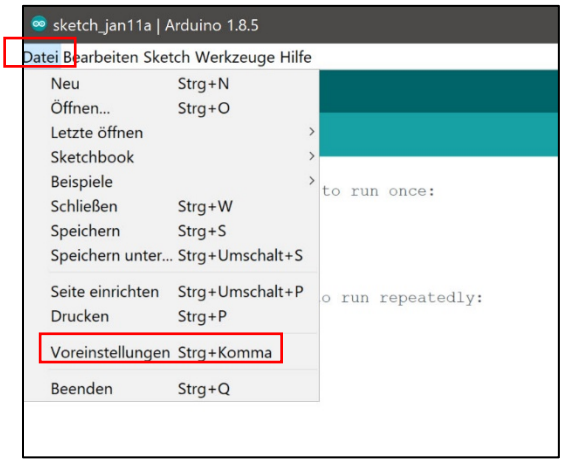
# Software Arduino für den Einsatz vorbereiten

Auf dem Desktop befindet sich die Verknüpfung mit dem Programm Arduino. 

Mit Klick auf das Symbol öffnet sich diese.

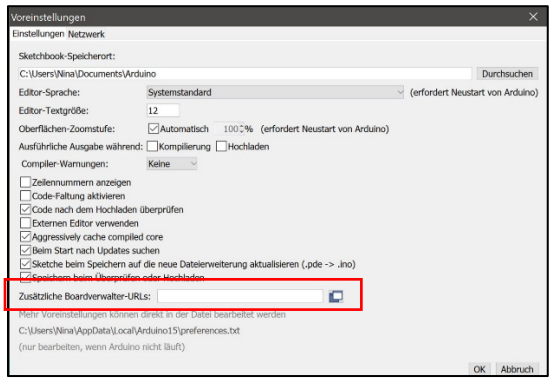


Auf **Datei** und dann **Voreinstellungen** klicken.



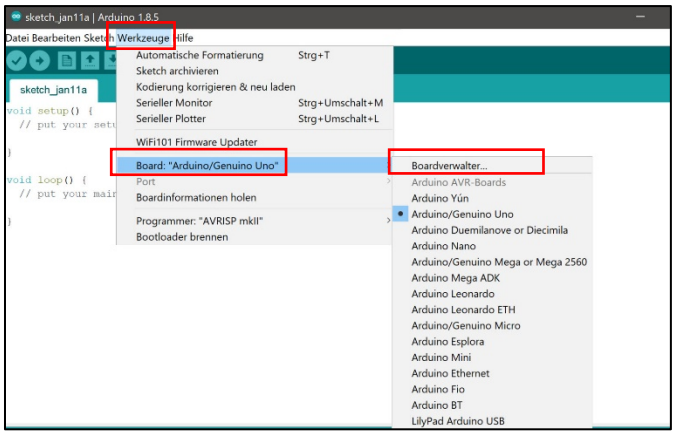

Hinter **Zusätzliche Boardverwalter-URLs:** in das Eingabefeld folgendes einfügen

http://arduino.esp8266.com/versions/2.3.0/package_esp8266com_index.json



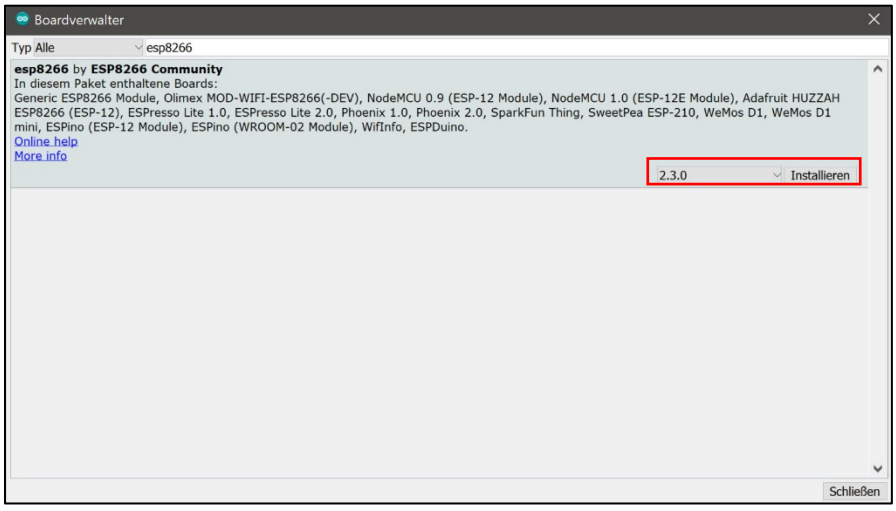

Mit **ok** bestätigen.

Dann auf **Werkzeuge**, dann **Board** und dann auf **Boardverwalter...** klicken



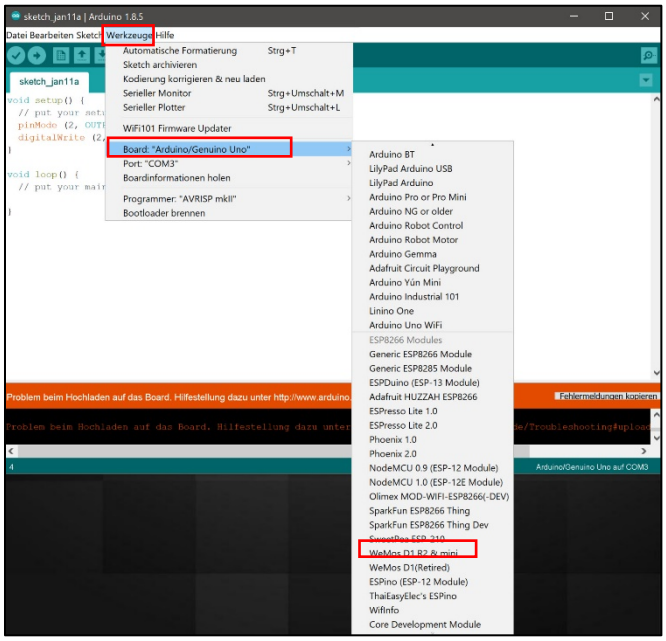

In die Suchzeile **esp8266** eingeben. Der Screenshot zeigt bereits das Suchergebnis.


Mit Klick auf das Suchergebnis erscheint rechts unten ein Feld mit einer Versionsnummer.



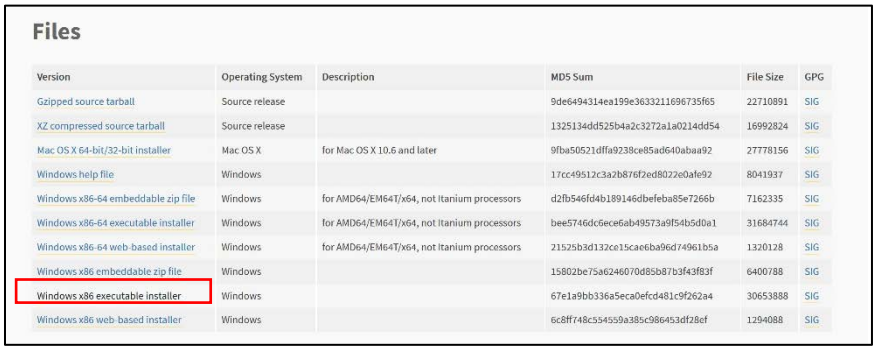

Die Version 2.3.0 des Suchergebnisses installieren und das Fenster schließen.

Dann auf **Werkzeuge**, dann **Board** und dann auf **Boardverwalter...** klicken und den **WeMos D1 R2 & mini auswählen**.



# Software Python installieren

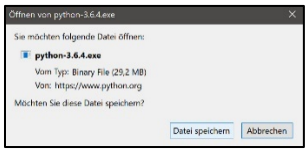
Damit Programme per OTA (Over The Air) hochgeladen werden können, muss Python installiert werden. Auf der Webseite https://www.python.org/downloads/release/python-364/ **Windows x86 executable installer** klicken.



**Files**

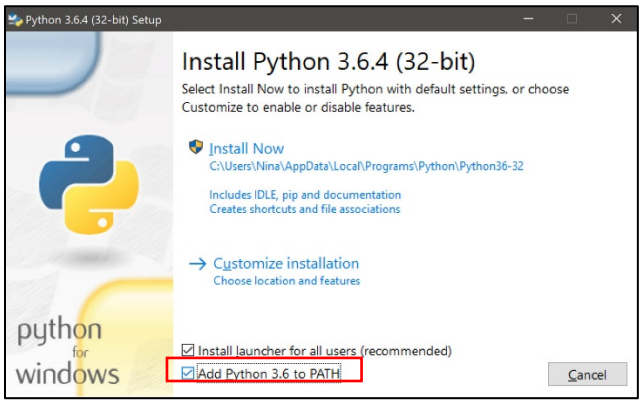
| Version | Operating System | Description | MD5 Sum | File Size | GPG |
|---|---|---|---|---|---|
| Gzipped source tarball | Source release | | 9de6494314ea199e3633211696735f65 | 22710891 | SIG |
| XZ compressed source tarball | Source release | | 1325134dd525b4a2c3272a1a0214dd54 | 16992824 | SIG |
| Mac OS X 64-bit/32-bit installer | Mac OS X | for Mac OS X 10.6 and later | 9fba50521dffa9238ce85ad640abaa92 | 27778156 | SIG |
| Windows help file | Windows | | 17cc49512c3a2b876f2ed8022e0afe92 | 8041937 | SIG |
| Windows x86-64 embeddable zip file | Windows | for AMD64/EM64T/x64, not Itanium processors | d2fb546fd4b189146dbefeba85e7266b | 7162335 | SIG |
| Windows x86-64 executable installer | Windows | for AMD64/EM64T/x64, not Itanium processors | bee5746dc6ece6ab49573a9f54b5d0a1 | 31684744 | SIG |
| Windows x86-64 web-based installer | Windows | for AMD64/EM64T/x64, not Itanium processors | 21525b3d132ce15cae6ba96d74961b5a | 1320128 | SIG |
| Windows x86 embeddable zip file | Windows | | 15802be75a6246070d85b87b3f43f83f | 6400788 | SIG |
| Windows x86 executable installer | Windows | | 67e1a9bb336a5eca0efcd481c9f262a4 | 30653888 | SIG |
| Windows x86 web-based installer | Windows | | 6c8ff748c554559a385c986453df28ef | 1294088 | SIG |

Das Programm wird wieder in dem voreingestellten Ordner gespeichert.

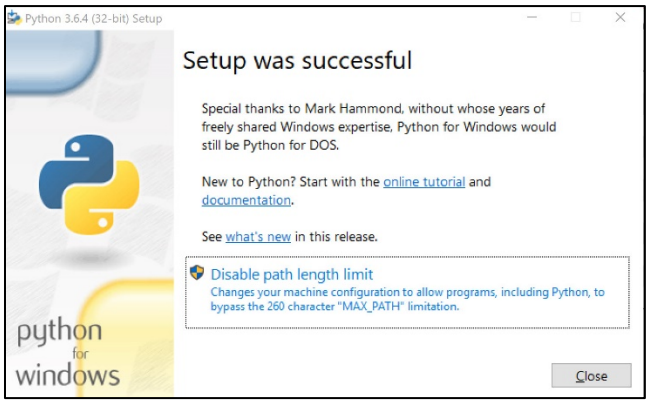



Mit Klick auf das Symbol das heruntergeladene Programm ausführen. 


Den Haken bei "Add Python 3.6 to PATH" setzen und auf **Install Now** klicken



Das folgende Fenster mit **Close** schließen.




Da mit hoher Wahrscheinlichkeit nichts im Programm getan wird, ohne dass man es auch speichern möchte, wird nun ein Ordner auf dem Rechner (z.B. auf dem Desktop) angelegt, der eindeutig benannt wird. Am besten ein Überordner Arduino mit Unterordnern für einzelne Projekte - im Zweifelsfall heißt das erste Projekt „Test1“. Gespeichert werden die Dateien voreingestellt unter

sketch_“monat“„Tag“„laufendeNummer“

# Software Arduino: Werkzeuge

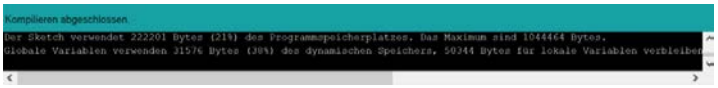
Die Menuleiste des Programms ist übersichtlich. Die fünf großen, runden bzw. quadratischen Symbole werden am häufigsten gebraucht. 

Beim Mouse-over wird das Symbol weiß und rechts der Symbole wird die Funktion beschrieben.



Beim Klick auf das Symbol wird die Tätigkeit ausgelöst. Beim **Haken** wird eine Überprüfung des programmierten Quellcodes durchgeführt. 

In der Kommandozeile unten wird der Vorgang als **Sketch wird kompiliert** dargestellt und auch der Abschluss des Vorgangs angezeigt.




Ein Kompiler (auch Compiler) ist ein Programm, das den Quellcode so übersetzt, dass unser WeMos ihn versteht. Deswegen war es auch wichtig, dass wir zuvor unter **Werkzeuge** und **Boards** unsere Ausgabe als **WeMos D1 mini** festgelegt haben.

Mit dem **Pfeil nach rechts** wird der programmierte Code auf den WeMos übertragen. Wenn man auf **Hochladen** klickt, wird automatisch gespeichert, kompiliert und dann hochgeladen. Sozusagen drei in eins. 

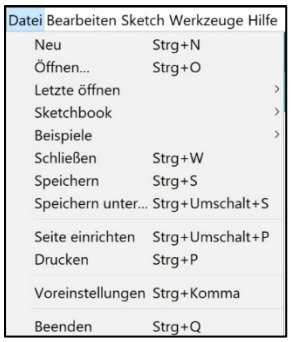

Mit dem **Datenblatt** kann ein neues Dokument erstellt werden. 


Der **Pfeil nach oben** führt zur Suche nach einem bereits bestehenden Dokument.



Mit dem **Pfeil nach unten** wird das aktuelle Dokument gespeichert. 

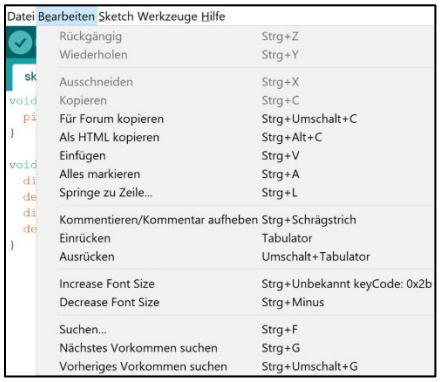
Das Menu führt zu Unterpunkten. Unter **Datei** verbergen sich die in vielen Programmen üblichen Befehle für **Speichern**, **Voreinstellungen**, etc. Es sind auch jeweils die Shortcuts angegeben, aber die Symbole wie oben beschrieben haben die gleichen Funktionen und sind mit weniger Klicks erreichbar.




Es ist sinnvoll, einmal alle Register aufzumachen. Dann hat man schnelle eine Ahnung wo man suchen soll, falls man mal etwas braucht.

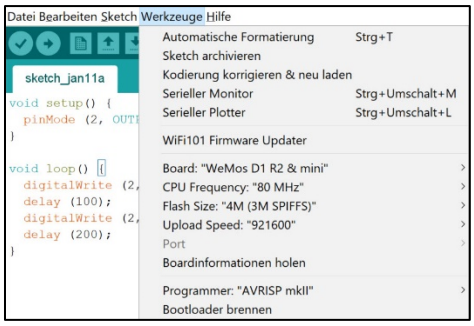
Aus dem Menupunkt **Bearbeiten** sei hier nur erwähnt, dass man mit dem **Tabulator** einrückt. Dazu muss man nicht das Menu öffnen, aber den **Tabulator** braucht man später in fast jeder Zeile.

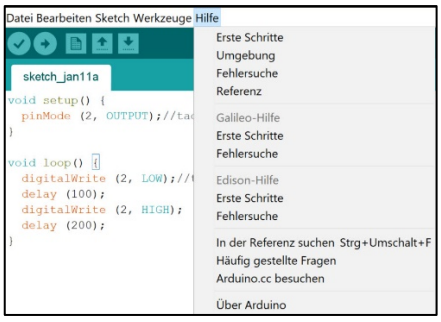
**F6** für die Suchfunktion könnte später auch mal interessant werden.




Der Menupunkt **Sketch** sei an dieser Stelle der Vollständigkeit halber erwähnt.

Die Menupunkte **Werkzeuge** und **Hilfe** einmal anschauen, zumachen und erst mal ad acta legen!

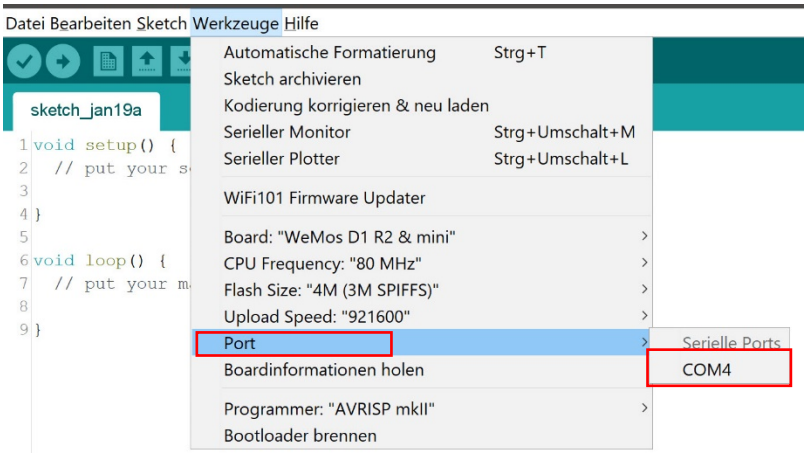



# WeMos mit dem Rechner verbinden

Benötigte Hardware: Board mit WeMos D1 mini (ist bereits verbunden, siehe oben), USB-Kabel, Rechner mit Software (siehe oben). Den WeMos über das USB-Kabel mit dem Rechner verbinden.

Jetzt muss erst der richtige **Port** (Hardwareschnittstelle) ausgewählt werden. Dazu geht man auf **Werkzeuge** und **Port** und wählt den entsprechenden Port aus. Sollten mehrere **COM**s zur Auswahl stehen kann man das USB-Kabel kurz entfernen. Der **COM**, der nun nicht mehr erscheint, wäre der richtige gewesen. Also wird das USB-Kabel wieder angesteckt und dieser **COM** ausgewählt.



# Programmierung

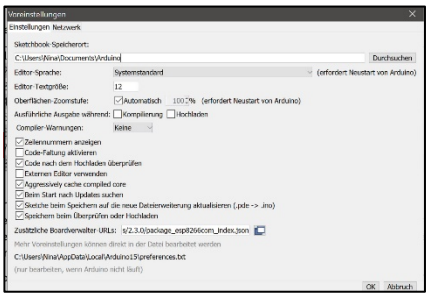
Unser WeMos D1 mini wird mit der Software Arduino angesteuert. Vorweg sei erwähnt, dass ein Programmcode (auch Quellcode genannt) immer englisch ist. Kommentare können auf Deutsch verfasst werden.

Der voreingestellte Programmcode ist leer und sieht so aus:

```
1    void setup() {
2        // put your setup code here, to run once:
3    }
4
5    void loop() {
6        // put your main code here, to run repeatedly:
7    }
8
```

An der linken Seite stehen die Zeilennummern (Line of Code oder kurz LOC). Diese werden automatisch vergeben. Hier sind dann auch alle Leerzeilen zur Strukturierung und Kommentarzeilen enthalten. Die Zeilenanzahl sagt also nichts über den Umfang des Programms aus. Die SLOC (Source Lines of Code) sind dagegen reine Programmierzeilen.

Sollten die Zeilennummern nicht erscheinen, kann man diese über **Voreinstellungen** und Haken bei **Zeilennummern anzeigen** anschalten.




Abgesehen von den noch unbekannten Worten fallen die Klammern auf. Superwichtig beim Programmieren ist es, nie eine Klammer zu vergessen. Das ist der wohl häufigste Fehler, weshalb ein Programm nicht funktioniert. Hier sieht das noch übersichtlich aus, aber bei umfangreichen Programmen passiert es schnell, dass man eine Klammer vergisst.

Es gibt zwei Methodenblöcke.

Beim void setup (void = leer, setup = Aufbau) werden Grundeinstellungen festgelegt. Hier wird z.B. festgelegt, ob ein Kanal ein In- oder ein Outputkanal ist. Dies wird nur einmal beim Programmstart ausgeführt und danach nie wieder.

Void ist ein Schlüsselwort und wird automatisch blau geschrieben. Setup ist eine Funktion und wird olivfarben dargestellt.

Ein void loop (void = leer, loop = Wiederholung) wird das Folgende ständig wiederholt. Hier wird also der eigentliche Programmablauf geschrieben.

Kommentare werden mit zwei ***//*** eingeleitet. Das sagt dem Programm nachher, dass alles was danach kommt für die Ausführung des Programms irrelevant ist. Kommentare werden automatisch in einem grau-lila geschrieben. Man kann einen Kommentar in die gleiche Zeile wie einen Befehl schreiben (mit Tabulator eingerückt natürlich), oder in eine neue Zeile. Die beiden beim Start immer vorhandenen Kommentarzeilen kann man löschen. Sie bieten dem Einsteiger eine Hilfe, wofür die einzelnen Blöcke stehen.

Man selbst sollte sich eine gute Kommentierung angewöhnen. Nicht zu viel, aber auch nicht zu wenig. Das richtige Maß ist sehr individuell und entwickelt sich im Laufe der weiteren Arbeit mit Arduino.

# On-board-LED

## On-board-LED anschalten

Arduino Programm starten und folgenden Code eingeben:

```
1    void setup() {
2        pinMode (2, OUTPUT);        // 2 für den on board Pinn, Output = Ausgang
3        digitalWrite (2, LOW);      // LED anschalten
4    }
5
6    void loop() {
7
8    }
```

Da die LED nur ein einziges Mal angehen soll, werden die Befehle unter void setup geschrieben.

Dass die On-board-LED eines WeMos D1 mini die Pinnnummer 2 hat, muss man zuvor aus einer Liste/Internet herausgesucht haben. Sie liegt bei jedem Board auf einer anderen Nummer.

Ein wenig außergewöhnlich ist das An- und Ausschalten der On-board-LED. Andere LEDs werden mit HIGH angeschaltet. Die On-board-LED schaltet beim WeMos aber umgekehrt. Deshalb wird hier zum Anschalten LOW verwendet.

Alles innerhalb der geschweiften Klammer wird mit dem Tabulator eingerückt. Es ist wichtig, sich an diese Kleinigkeiten zu halten, da sonst bei längeren Programmen der Überblick verloren geht. Nur ein ordentlicher Programmierer ist ein guter Programmierer.

Der Befehl wird automatisch in rot geschrieben.

An jedem Befehlsende steht ein Semikolon. Ein vergessenes Semikolon ist der wohl zweithäufigste Fehler, wenn ein Programm nicht funktioniert.

Man schreibt alle Funktions- und Variablennamen klein. Beim zweiten Wort wird der erste Buchstabe großgeschrieben.

Jetzt wird der WeMos mit dem USB-Kabel an den Rechner angeschlossen. Die LED leuchtet einmal zur Kontrolle blau auf.

Die Datei muss nun zuerst gespeichert und dann an den WeMos übergeben werden (Symbol Haken und dann Symbol Pfeil).

## On-board LED ausschalten

```
1    void setup() {
2        pinMode (2, OUTPUT);        // 2 für den on board Pinn, Output = Ausgang
3        digitalWrite (2, HIGH);     // LED ausschalten
4    }
5
6    void loop() {
7
8    }
```

# On-board-LED blinken lassen

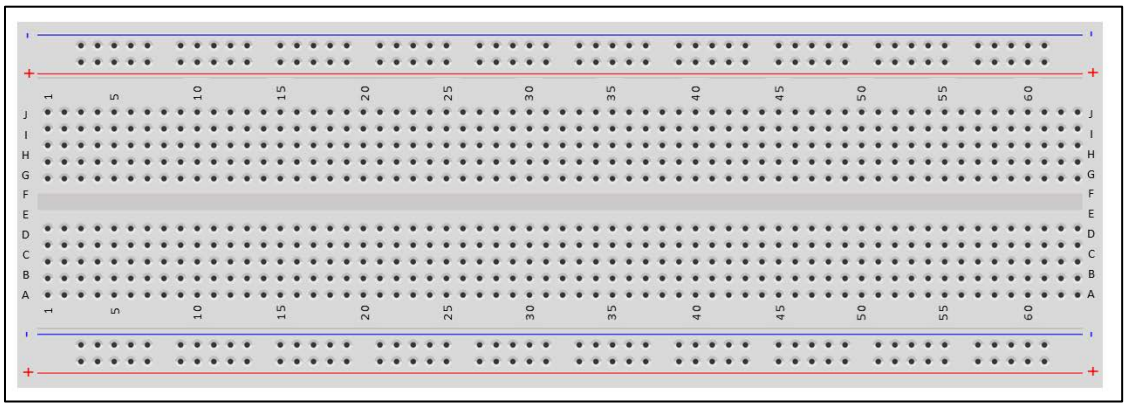
Der Befehl pinMode bleibt wo er ist, denn er definiert ja nur den Pinn als Ausgang. digitalWrite wird von void setup (einmal ausführen) nach void loop (immer wiederholen) kopiert. Es werden Zeiten in Millisekunden eingegeben, bis der nächste Befehl ausgeführt wird.

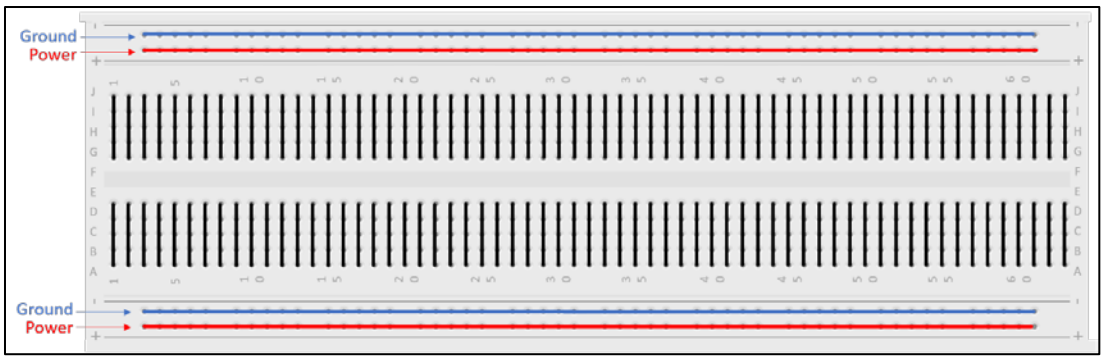
```
1   void setup() {
2       pinMode (2, OUTPUT);        // 2 für den on board Pinn, Output = Ausgang
3   }
4
5   void loop() {
6       digitalWrite (2, LOW);      // LED anschalten
7       delay (100);                // warte 100ms
8       digitalWrite (2, HIGH);     // LED ausschalten
9       delay (200);                // warte 200ms
10  }
11
```

Der USB-Stecker kann jetzt entfernt werden. Da der WeMos keinen Strom bekommt, hört die LED auf zu blinken. Wenn man nun das Netzteil ansteckt, fängt die LED wieder an zu blinken, da das Programm ja noch auf dem WeMos gespeichert ist.

# Das Breadboard

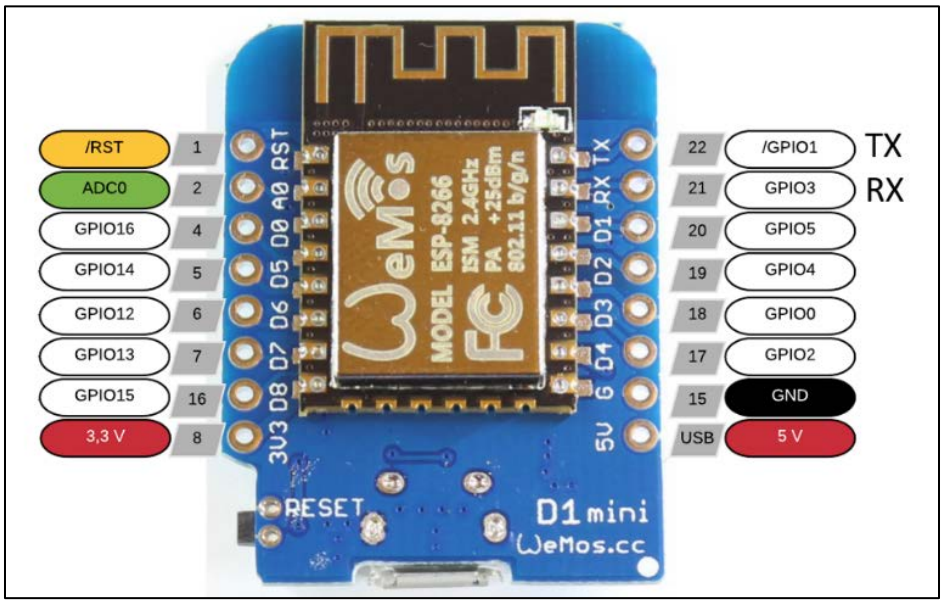
In der Kunststoffplatte befinden sich Kontaktfedern. Die Kontakte sind in zwei parallelen Reihen a 5 Steckplätze geordnet und mit 1 bis 60 nummeriert. Am Rand gibt es zwei Querreihen zwischen einer blauen und einer roten Linie. Blau ist Ground (die Erdung); rot ist Power.





# WeMos

Aufbau:




Die Stromversorgung erfolgt über den Micro-USB-Anschluss. Dieser ist zusätzlich die serielle Schnittstelle zum PC. Ein interner Spannungsregler erzeugt aus den 5V die erforderlichen 3,3 Volt. Wenn das Netzteil hier angeschlossen wird, werden 5V in 3,3V umgewandelt.

Achtung: der analoge, der RST und alle digitalen PINs des WeMos D1 mini sind 3,3 Volt! Andere Arduinos verwenden 5V.

Ground ist die Erdung.

GPIO (general purpose input/output) sind Allzweckeingaben bzw. – ausgaben. Sie haben keinen vorgegebenen Zweck und sind standardmäßig unbelegt.

ADC (analog-to-digital converter) wandelt ein analoges Eingangssignal in ein digitales Signal um.

RST bedeutet reset (zurücksetzen). Wenn dieser Pinn benutzt wird passiert das gleiche, wie wenn man den kleinen weißen Knopf drückt: das Programm wird neu gestartet.

TX ist serielles Senden und RX ist serielles Empfangen. Es kann so eine höhere Taktfrequenz genutzt werden, um die Übertragungsrate zu erhöhen.

# Betrieb über das Netzteil (Achtung Sicherheitsrelevant)

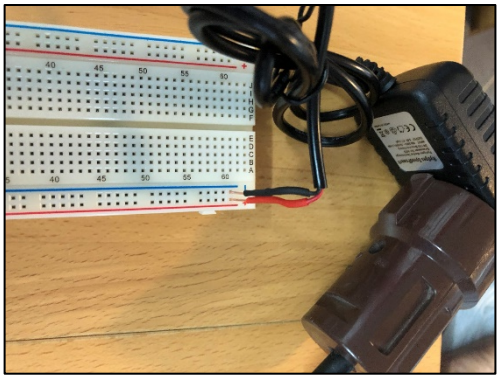
Man darf nur entweder das Netzteil oder das USB-Kabel anschließen!!! Also bitte immer erst USB trennen und dann das Netzteil anschließen.

Wenn man das Netzteil anschließt, immer zuerst die Pole verbinden, dann den Netzstecker einstecken.

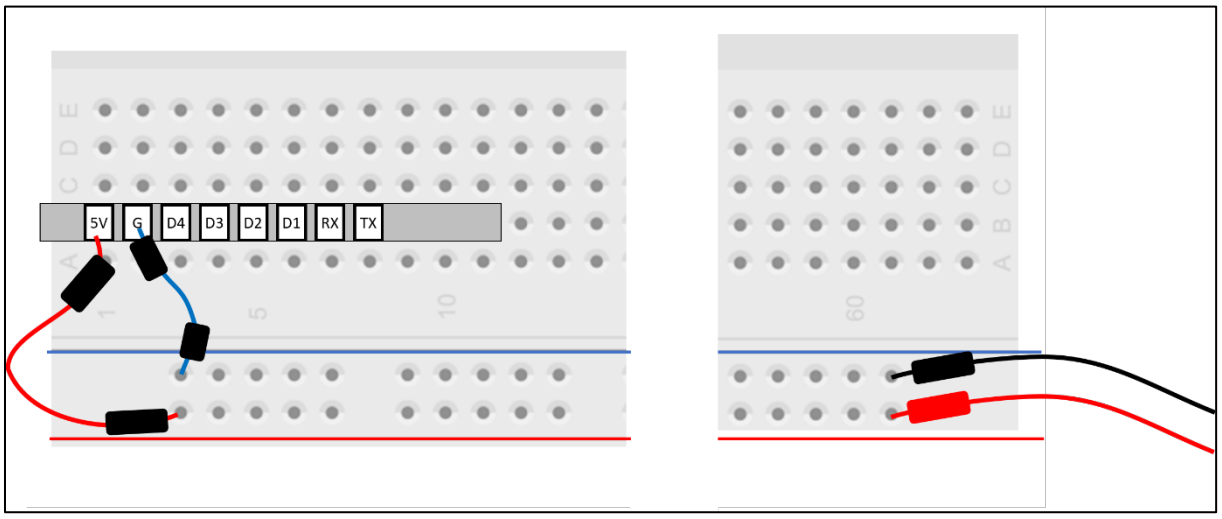
Wenn man das Netzteil abmacht, immer zuerst das Netzstecker ziehen, ein bisschen warten (damit die Restladungen abgeflossen sind) und dann die Pole abmachen.

Die Pole liegen sehr nah beieinander und könnten, wenn sie sich unter Strom berühren, einen Kurzschluss auslösen.

Das Breadboard bzw. der WeMos wird mit zwei Jumperwire verbunden.



Das Netzteil wird über das schwarze Kabel mit Minus Ground) und das roten Kabel mit Plus (Power) verbunden (siehe Breadboard). Dann liegen hier 5V an!!!



# Grundsätzliches Vorgehen bei externen Schaltungen

1. Programmierung
2. Programm auf den WeMos laden
3. USB-Kabel lösen
4. Externe Schaltung stecken
5. prüfen
6. Pole des Netzgeräts anschließen
7. Netzstecker einstecken

Wenn es nicht funktioniert könnte es an der Hardware oder der Software liegen

## Fehlersuche Hardware

1. Netzstecker lösen
2. Schaltung prüfen, ggf. ändern
3. Netzstecker anschließen

## Fehlersuche Software

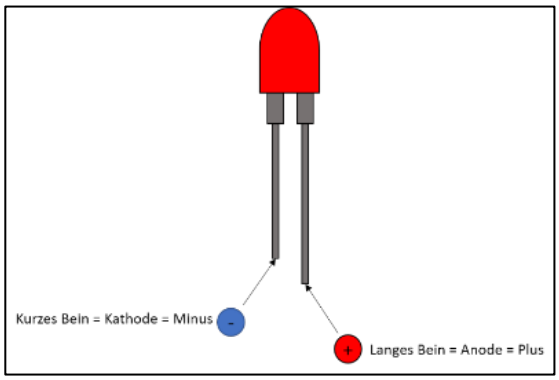
1. Netzstecker lösen
2. Externe Schaltung vom WeMos lösen (meist genügt es, das Ground- und 5V-kabel zu lösen)
3. Programmierung prüfen und ändern
4. USB anschließen
5. Programm überspielen
6. USB-Kabel lösen
7. Externe Schaltung in den WeMos einstecken
8. Netzstecker anschließen

# Einzelne externe LED

## Externe LED anschließen

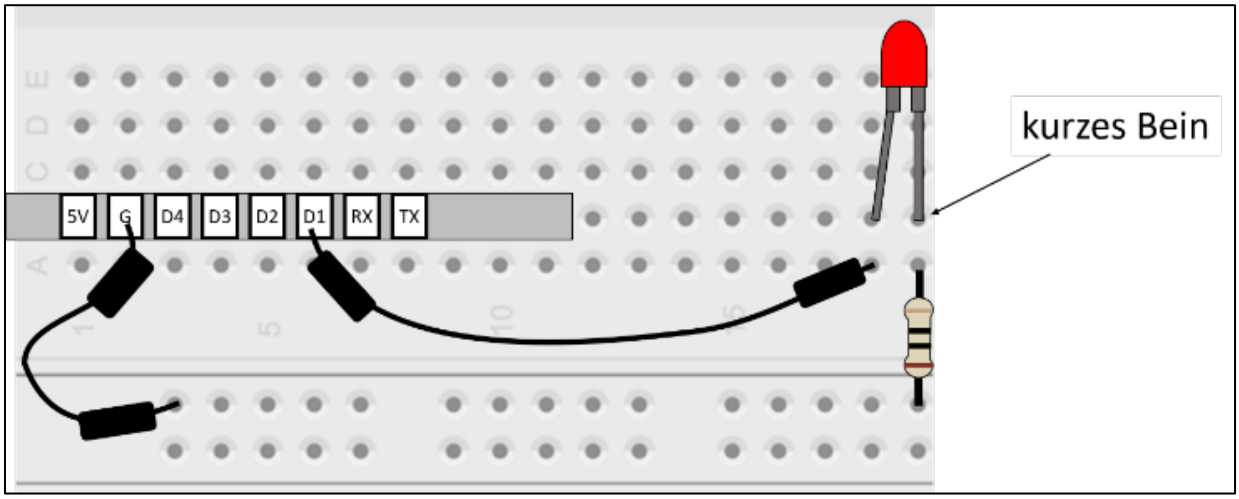
Dazu braucht man eine LED und einen passenden Vorwiderstand (330 Ohm).

Der Widerstand wird mit Ground und einer Kontaktreihe verbunden. Die LED wird mit dem kurzen Bein mit der gleichen Kontaktreihe verbunden.




Das lange Bein wird mit der nebenliegenden Kontaktreihe verbunden. Ein Jumperwire verbindet diese Kontaktreihe mit dem GPIO D1.

Ein weiteres Jumperwire verbindet G mit der Groundreihe.



## Externe LED anschalten

Die LED ist mit dem Ausgang D1 = GPIO 5 verbunden. Daher muss der pinMode 5 gewählt werden.

```
1    void setup(){
2        pinMode (5, OUTPUT);        // für die externe LED, Output = Ausgang
3        digitalWrite (5, HIGH);     // LED anschalten
4    }
5
6    void loop(){
7    }
```

## Externe LED ausschalten

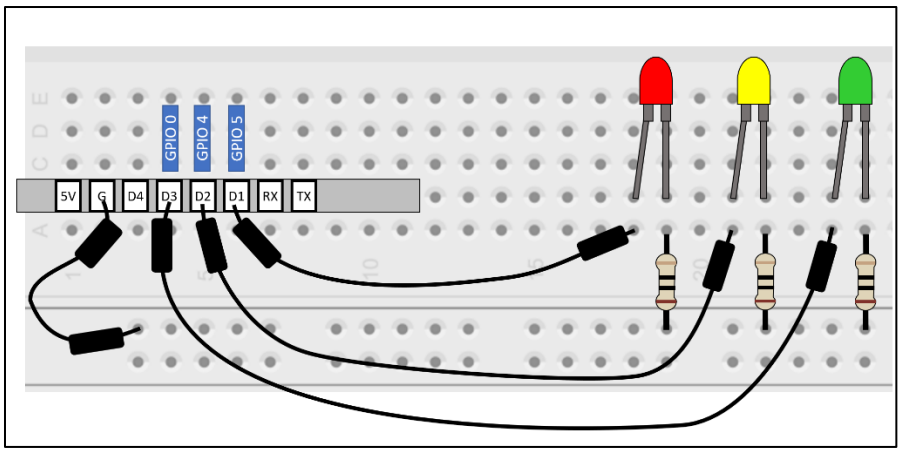
```
1    void setup(){
2        pinMode (5, OUTPUT);        // für die externe LED, Output = Ausgang
3        digitalWrite (5, LOW);      // LED ausschalten
4    }
5
6    void loop(){
7    }
```

## Externe LED blinken lassen

```
12   void setup(){
13       pinMode (5, OUTPUT);        // für die externe LED, Output = Ausgang
14   }
15
16   void loop(){
17       digitalWrite (5, LOW);      // LED ausschalten
18       delay (200);                // warte 200ms
19       digitalWrite (5, HIGH);     // LED anschalten
20       delay (200);                // warte 200ms
21   }
```

# mehrere externe LEDs

Der Einfachheit halber werden 3 LEDs in Ampelfarben (rot, gelb, grün) plus passende Vorwiderstände gewählt.




```
1   void setup(){
2       pinMode (5, OUTPUT);        // für die rote LED, Output = Ausgang
3       digitalWrite (5, HIGH);     // LED anschalten
4       pinMode (4, OUTPUT);        // für die gelbe LED, Output = Ausgang
5       digitalWrite (4, HIGH);     // LED anschalten
6       pinMode (0, OUTPUT);        // für die grüne LED, Output = Ausgang
7       digitalWrite (0, HIGH);     // LED anschalten
8   }
9
10  void loop(){
11  }
```

Üblicherweise wird zunächst der pinMode für jeden Pin festgelegt und dann geschaltet. Also alle drei pinMode unter einander und dann alle drei digitalWrite unter einander.

```
1   void setup(){
2       pinMode (5, OUTPUT);        // für die rote LED, Output = Ausgang
3       pinMode (4, OUTPUT);        // für die gelbe LED, Output = Ausgang
4       pinMode (0, OUTPUT);        // für die grüne LED, Output = Ausgang
5       digitalWrite (5, HIGH);     // LED anschalten
6       digitalWrite (4, HIGH);     // LED anschalten
7       digitalWrite (0, HIGH);     // LED anschalten
8   }
9
10  void loop(){
11  }
```

# Funktion define

Es ist gerade bei komplexen Programmcodes einfacher, gleich zu Beginn Dinge zu definieren. Dadurch muss man nicht jedes Mal überlegen, was wo angeschlossen ist, sondern man nutzt einfach einen definierten Begriff.
Diese Definition erfolgt vor dem void setup, da sie Global für den gesamten Code gilt.
Wir legen fest, dass der Ausgang **5** der **Pin** mit der **rot**en LED, der Ausgang **4** der Pin mit der **gelb**en LED und der Ausgang **0** der Pin mit der **grün**en LED ist. Variablen bekommen grundsätzlich immer englische Namen.

```
1   #define pinRed 5
2   #define pinYellow 4
3   #define pinGreen 0
4
5   void setup(){
6       pinMode (pinRed, OUTPUT);          // für die rote LED, Output = Ausgang
7       pinMode (pinYellow, OUTPUT);       // für die gelbe LED, Output = Ausgang
8       pinMode (pinGreen, OUTPUT);        // für die grüne LED, Output = Ausgang
9       digitalWrite (pinRed, HIGH);       // LED anschalten
10      digitalWrite (pinYellow, HIGH);    // LED anschalten
11      digitalWrite (pinGreen, HIGH);     // LED anschalten
12  }
13
14  void loop(){
15  }
16
```

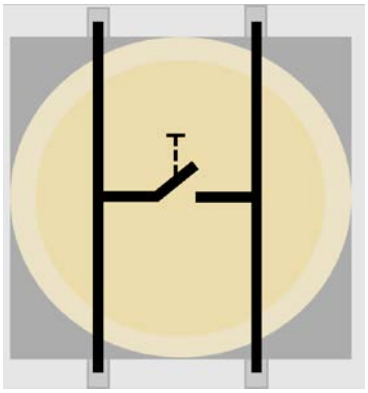
# Ampelschaltung

Mit den Kenntnissen der vorhergehenden Abschnitte kann eine Ampelschaltung erstellt werden.

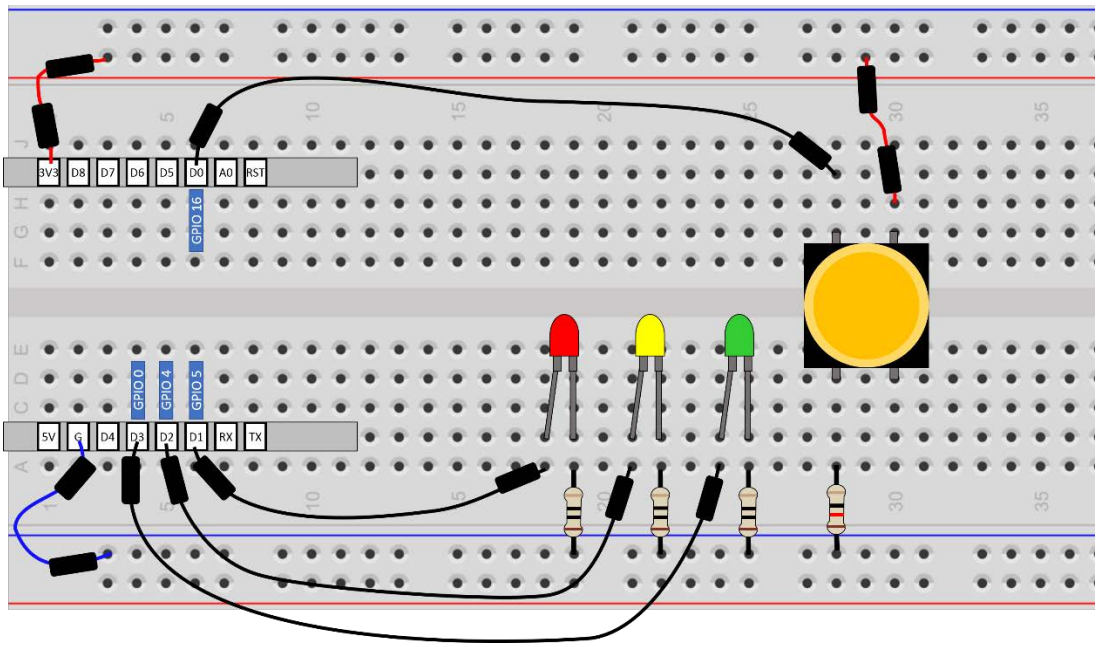
```
1   #define pinRed 5
2   #define pinYellow 4
3   #define pinGreen 0
4
5   void setup(){
6       pinMode(pinRed, OUTPUT);          // für die rote LED, Output = Ausgang
7       pinMode(pinYellow, OUTPUT);       // für die gelbe LED, Output = Ausgang
8       pinMode(pinGreen, OUTPUT);        // für die grüne LED, Output = Ausgang
9   }
10
11  void loop(){
12      digitalWrite (pinRed, HIGH);      // rote LED anschalten
13      delay (2000);                     // warte 2000ms
14      digitalWrite (pinYellow, HIGH);   // gelbe LED anschalten
15      delay (500);                      // warte 500ms
16      digitalWrite (pinRed, LOW);       // rote LED ausschalten
17      digitalWrite (pinYellow, LOW);    // gelbe LED ausschalten
18      digitalWrite (pinGreen, HIGH);    // grüne LED anschalten
19      delay (2000);                     // warte 2000ms
20      digitalWrite (pinGreen, LOW);     // grüne LED ausschalten
21      digitalWrite (pinYellow, HIGH);   // gelbe LED anschalten
22      delay (500);                      // warte 500ms
23      digitalWrite (pinYellow, LOW);    // gelbe LED ausschalten
24  }
25
```

# Ampel mit Knopf

Ein Knopf hat vier Pins. Es liegen immer zwei nebeneinander. Wenn der Knopf gedrückt wird, sind die nebeneinanderliegenden Pins verbunden - sonst nicht. Gegenüberliegende Pins sind immer verbunden. In diesem Schaltbild ist es genau dargestellt:



Zunächst wird ein Knopf in der Schaltung wie folgt hinzugefügt.




Hier ist der Datenpin D0 mit dem 1KΩ Widerstand verbunden. Die andere Seite des Widerstandes ist mit GND verbunden. Wenn der Knopf gedrückt wird, ist der Datenpin direkt mit 3,3 Volt verbunden. Wenn er nicht gedrückt ist, liegen 0 Volt an, denn der Datenpin wird über den Widerstand auf Ground gezogen. Deshalb wird ein so eingebauter Widerstand auch „Pull down-Widerstand“ genannt.

```
1   #define pinRed 5
2   #define pinYellow 4
3   #define pinGreen 0
4   #define pinButton 16
5
6   bool buttonState=false;
7
8   void setup(){
9       pinMode(pinRed, OUTPUT);              //für die rote LED, Output = Ausgang
10      pinMode(pinYellow, OUTPUT);           //für die gelbe LED, Output = Ausgang
11      pinMode(pinGreen, OUTPUT);            //für die grüne LED, Output = Ausgang
12      pinMode(pinButton, INPUT);            //den Knopf als INPUT = Eingang festlegen
13      digitalWrite(pinRed, HIGH);           //Startwert für die Rote LED festlegen
14      //Da die Ampel aus wäre bis der Knopf zum ersten Mal gedrückt wird,
15      //muss sie am Anfang einmal an gehen
16  }
17
18  void loop(){
19      buttonState=digitalRead(pinButton);   //Knopf auslesen, in Variable schreiben
20      if(buttonState==true){                //Variable prüfen
21          digitalWrite(pinYellow, HIGH);    //gelbe LED anschalten
22          delay(500);                       //warte 500ms
23          digitalWrite(pinRed, LOW);        //rote LED ausschalten
24          digitalWrite(pinYellow, LOW);     //gelbe LED ausschalten
25          digitalWrite(pinGreen, HIGH);     //grüne LED anschalten
26          delay(2000);                      //warte 2000ms
27          digitalWrite(pinGreen, LOW);      //grüne LED ausschalten
28          digitalWrite(pinYellow, HIGH);    //gelbe LED anschalten
29          delay(500);                       //warte 500ms
30          digitalWrite(pinYellow, LOW);     //gelbe LED ausschalten
31          digitalWrite(pinRed, HIGH);       //rote LED anschalten
32          delay(2000);                      //warte 2000ms
33      }
34  }
```

# Syntaxliste

Syntax kennt man aus der Grammatik von natürlichen Sprachen als Satzbau. Hier wird das Zusammenfügen von Wörtern zu größeren Einheiten bis hin zu Sätzen festgelegt.

In einer formalen Sprache, also zum Beispiel in der Programmiersprache, ist Syntax ein System von Regeln, nach denen Ausdrücke, Formeln, Programmtexte oder andere Texte gebildet werden.

## Funktionen

```
pinMode(pin, mode);
```

Die Funktion pinmode legt den Modus des Pins als Input oder Output fest. Sie muss mindestens einmal aufgerufen werden, bevor ein Wert an dem angegebenen Pin ausgelesen oder geschrieben werden kann.

pin: Ganzzahliger Wert der Nummer des Pins, der konfiguriert werden soll.
mode: INPUT um Werte auslesen zu können oder OUTPUT um Werte schreiben zu können.

```
digitalRead(pin);
```

Die Funktion digitalRead liest den Wert des digitalen Pins aus und gibt einen boolschen Wert (true oder false) zurück.

pin: Ganzzahliger Wert der Nummer des digitalen Pins, der ausgelesen werden soll.

```
digitalWrite(pin, value);
```

Die Funktion digitalWrite schreibt einen Wert auf den angegebenen Pin. Wenn LOW geschrieben wird, liegen 0 Volt an. Wenn HIGH geschrieben wird, liegen 3,3 Volt an.
(Achtung! Bei andern Arduino Boards können statt 3,3 Volt 5 Volt anliegen.)

pin: Ganzzahliger Wert der Nummer des Pins, der beschrieben werden soll.
value: Wert mit dem der Pin beschrieben werden soll. 1, true und HIGH bedeuten das Gleiche, üblicher Weise wird jedoch HIGH genutzt. 0, false und LOW bedeuten das gleiche, üblicher Weise wird jedoch LOW genutzt.

```
analogRead(pin, value);
```

Die Funktion analogRead liest den Wert des analogen Pins aus und gibt einen int-Wert zurück. Der zurückgegebene Wert hängt von der anliegenden Spannung ab und liegt zwischen 0 und 1023 [0V bis 3,3V].

Achtung! Wenn mehr als 3,3V anliegen könnte der Arduino Schaden nehmen.

pin: Ganzzahliger Wert der Nummer des analogen Pins, der ausgelesen werden soll.

```
delay(t);
```

Die Funktion delay lässt das gesamte Programm warten. Während gewartet wird kann nichts anderes gemacht werden, das Programm wird unterbrochen und nach der Wartezeit wider fortgesetzt.

t: Zu wartende Zeit in Millisekunden (1000 Millisekunden = 1 Sekunde).

```
millis();
```

Die Funktion millis gibt die Anzahl an Millisekunden zurück, die seit Programmstart vergangen sind. Sie gibt eine unsigned long zurück. Der Wert fängt etwa alle 50 Tage wieder bei 0 an, weil die Variable dann überläuft.

# Konstruktorbefehle

Diese Befehle werden einmal global festgelegt, also vor dem Setup, und gelten dann für das ganze Programm.

```
#define name value
```

name: ein beliebiger Name, den die Variable tragen soll.
wert: der Wert, den die Variable haben soll.

```
#include <library>
```

library: der Dateiname der Bibliothek, die eingebunden werden soll.

Beim Ansteuern von RGB LEDs wird zum Beispiel eine library (Bibliothek) genutzt, um das Programmieren zu vereinfachen. Ein ziemlich komplexer Code, der vom Hersteller geschrieben wurde, wird verwendet, damit man ihn nicht selbst schreiben muss.

# Variable

Eine **Variable** ist in der Mathematik ein Platzhalter, also eine Leerstelle in einem logischen oder mathematischen Ausdruck.

Eine **boolesche Variable** ist eine Variable, die nur zwei Zustände annehmen kann. Diese Zustände sind **true** und **false**. Diese Variablen können miteinander verknüpft werden und finden bei bedingten Anweisungen oder Schleifen Anwendung.

```
qualifier type name=value;
```

qualifier: optional. einer der Vorsätze, siehe unten.
type: einer der Datentypen, siehe unten.
name: ein beliebiger Name, den die Variable tragen soll.
value: der Startwert, den die Variable haben soll.

# Qualifier (Vorsätze)

const: Im gesamten Code unveränderbar, also konstant
static: Wird nur beim ersten Aufruf erstellt und auf den Startwert gesetzt.
Bei allen weiteren Aufrufen wird der Befehl ignoriert.

# Datatypes (Datentypen)

| Name | Bedeutung | Werte | Speicher | |
|---|---|---|---|---|
| bool | Wahrheitswert | false, true (0, 1) | 1 byte [8 bit] | Das gleiche wie boolean |
| boolean | Wahrheitswert | false, true (0, 1) | 1 byte [8 bit] | Das gleiche wie bool |
| int | Ganze Zahl | -32.768 bis 32.767<br>[-2^15 bis 2^15-1] | 2 bytes [16 bit] | |
| long | Ganze Zahl | -2.147.483.648 bis 2.147.483.647<br>[-2^31 bis 2^31-1] | 4 bytes [32 bit] | |
| unsigned int | Ganze Zahlen, kein Minus | 0 bis 65.535<br>[0 bis 2^16-1] | 2 bytes [16 bit] | |
| unsigned long | Ganze Zahlen, kein Minus | 0 bis 4.294.967.295<br>[0 bis 2^32-1] | 4 bytes [32 bit] | |
| float | Kommazahl | Ungefähr -3,4*10^38 bis 3,4*10^38 | 4 bytes [32 bit] | Nur 6-7 Stellen insgesamt |
| String | Kette von Symbolen | Alle ASCII Zeichen<br>Bsp.: „Hallo Welt-123“ | So viel, wie nötig | Kann Texte oder andere Ketten von Symbolen speichern. Ist der einzige Datentyp, der groß geschrieben wird. |

Bsp.:

```
int number=5;
bool buttonState=false;
static unsigned long timer=millis();
String ourHomepage=”www.physikusse.de”;
static int personCounter=0;
```

# Struktur und Operatoren

In der Mathematik sind Operatoren Vorschriften, wie zum Beispiel Pluszeichen + und geteilt-Zeichen. Mit Operatoren kann man also Berechnungen durchführen. Außerdem kann man mit ihnen Zuweisungen vornehmen, Vergleiche durchführen und anderes. Sie werden nach ihrem Typ unterschieden und nach der Anzahl ihrer Operanten. Operanten sind die Dinge, mit denen ein Operator etwas macht. Das + Zeichen hat beispielsweise 2 Operanten, nämlich einen Operanten vor dem Operator und einen danach, und mit denen führt der Operator eine Operation durch.

if prüft immer etwas auf Wahrheit.

else definiert, was gemacht werden soll, wenn if falsch ist.

```
1  if(x){
2          //Alles, was hier steht wird nur dann ausgeführt,
3          //wenn der Inhalt der Klammern (also x) wahr ist
4  }
5  else{
6          //Alles, was hier steht wird ausgeführt,
7          //wenn die Aussage x falsch ist. Das else ist optional,
8          //muss also nicht vorhanden sein.
9  }
```

**x:** **auf Wahrheit zu überprüfende Aussage**

**Bsp.:**

```
1  if(a>5)        //Wenn a größer als 5 ist
2  if(a+b==c)     //Wenn die Summe aus a und b gleich c ist
3  if(a!=b)       //Wenn a ungleich b ist
```

```
1  for(teil1; teil2; teil3){
2          //teil4
3  }
```

Schritt 1: Teil 1 wird ein einziges Mal zu Anfang ausgeführt.
Schritt 2: Wenn die Aussage in Teil 2 wahr ist, wird Teil 4 ausgeführt, sonst ist die for-Schleife beendet.
Schritt 3: Teil 3 wird immer nach Teil 4 ausgeführt und dann wird zu Schritt 2 gesprungen.

**Bsp.:**

```
4   for(int i=0; i<5; i++){
5           //Alles, was hier steht wird solange ausgeführt,
6           //wie i kleiner als 5 ist
7           //Nach jedem Durchgang wird i um eins größer
8           //(i++ macht das gleiche, wie i=i+1)
9   }
```

## Vergleichsoperatoren

In der if- und in der for-Schleife werden Vergleichsoperatoren verwendet, um Variablen miteinander zu vergleichen.

```
1   !=      //ungleich
2   ==      //gleich
3   >       //größer als
4   <       //kleiner als
5   >=      //größer oder gleich
6   <=      //kleiner oder gleich
```

## Arithmetische Operatoren

Arithmetik umfasst vor allem Grundrechenarten (Addition (Zusammenzählen), Subtraktion (Abziehen), Multiplikation (Vervielfachen), Division (Teilen) mit den natürlichen Zahlen.

Um zu rechnen und variablen Werte zuzuweisen werden arithmetische Operatoren verwendet.

```
1.  =               //Einen Wert zuweisen
2.  +-*/            //Plus Minus Mal Geteilt
3.  c++;            //++ addiert eins, -- subtrahiert eins
4.  a=b*(c-d)+e/f;  //Punkt vor Strich und Klammer vor Punkt wird beachtet
```